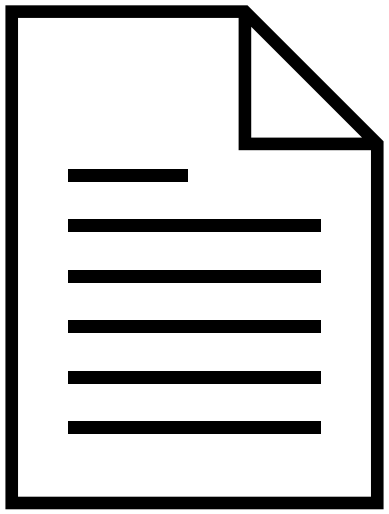
E-Tutoring

Programmazione per Dispositivi Mobili

Flutter Application

Bortolotti Simone, De Cenzo Davide, Marignati Luca

Sommario



1. Strumenti e librerie utilizzate
2. Metodologia agile
3. Architettura dell'applicazione
4. MySQL Database
5. Paradigma MVC
6. Funzionalità dell'applicazione (generali, tutor e studente)
7. Software: caratteristiche principali
8. Testing
9. Conclusioni

Strumenti e librerie utilizzate
File pubspec.yaml

- Framework: Flutter/Dart per il front-end;
- Web Services sviluppati in linguaggio PHP per l'interazione con il DB;
- MySQL Database per la memorizzazione dei dati;
- Librerie Flutter principali:
  - **Flutter_secure_storage** per il salvataggio delle credenziali dell'utente nel dispositivo mobile;
  - **Http**;
  - **Mockito** per il testing;
  - **flutter_local_notifications** per la gestione delle notifiche;
  - **flutter_localizations**
  - **intl**
  - …

```yaml
dependencies:
  flutter:
    sdk: flutter
  flutter_localizations:
    sdk: flutter
  font_awesome_flutter: ^8.8.1

  scrollable_positioned_list: ^0.1.7
  http: ^0.13.3

  # store & load data securely
  flutter_secure_storage: ^4.2.0

  intl: ^0.17.0

  move_to_background: ^1.0.2
```

Metodologia Agile

Caratteristiche:

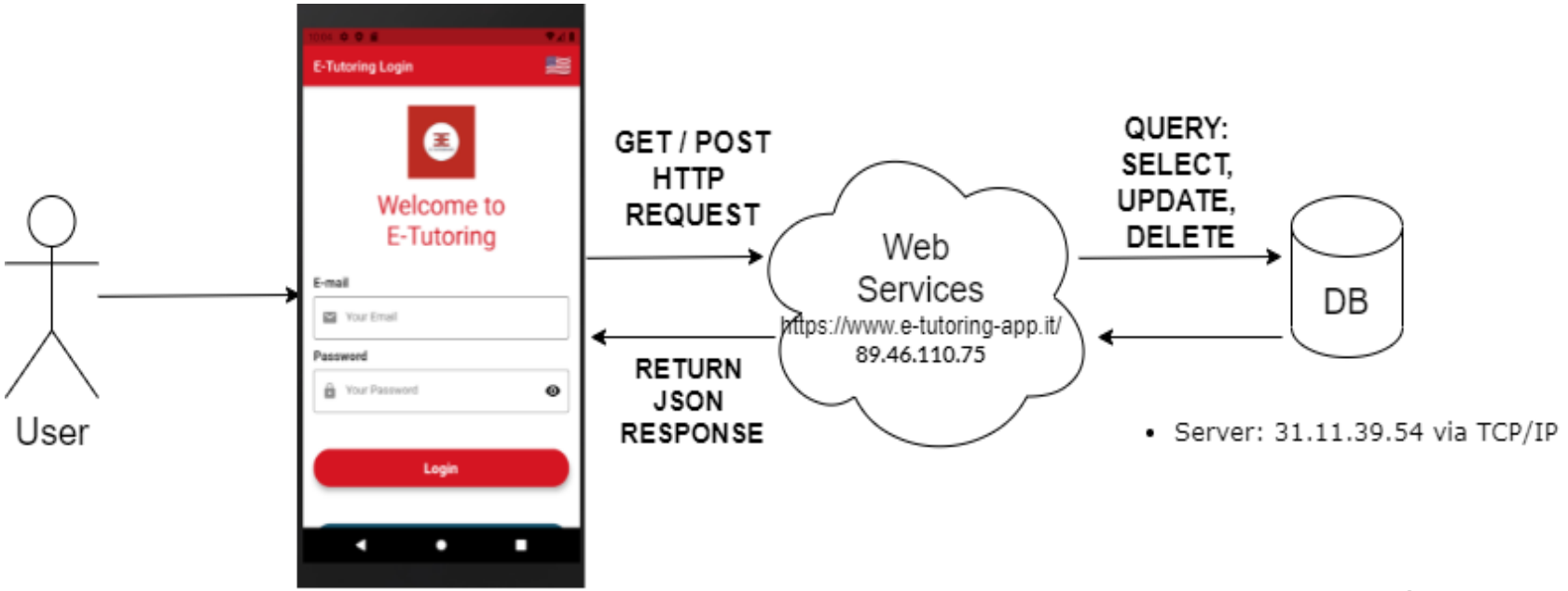
- Comunicazione
- Pair programming
- Semplicità
- Modifiche incrementali
- Refactoring del codice
- Simple Design
- Testing
- Feeback continuo

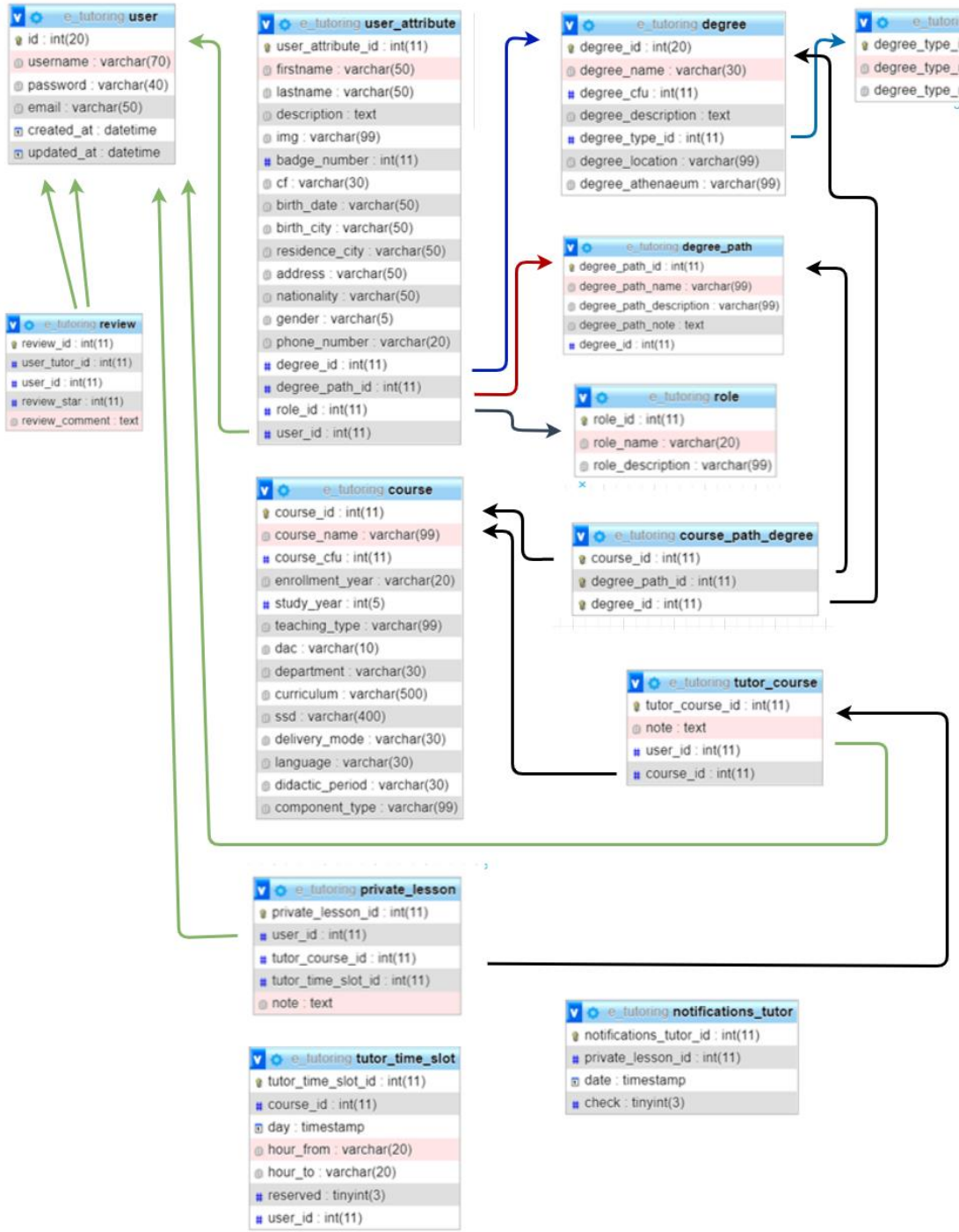
Architettura dell’applicazione

Architettura dell'applicazione

CLIENT → WEB SERVICES → DB MYSQL

- l'utente interagisce con l'applicazione tramite un client (dispositivo mobile Android o IOS);
- il client effettua chiamate http (GET/POST method) richiamando i web services implementati;
- i WS effettuano query (select) oppure operazioni di write (update o delete) sui dati contenuti nel DB MySQL;
- i WS restituiscono, in formato JSON, i dati richiesti oppure l'esito dell'operazione richiesta (successo o fallimento).





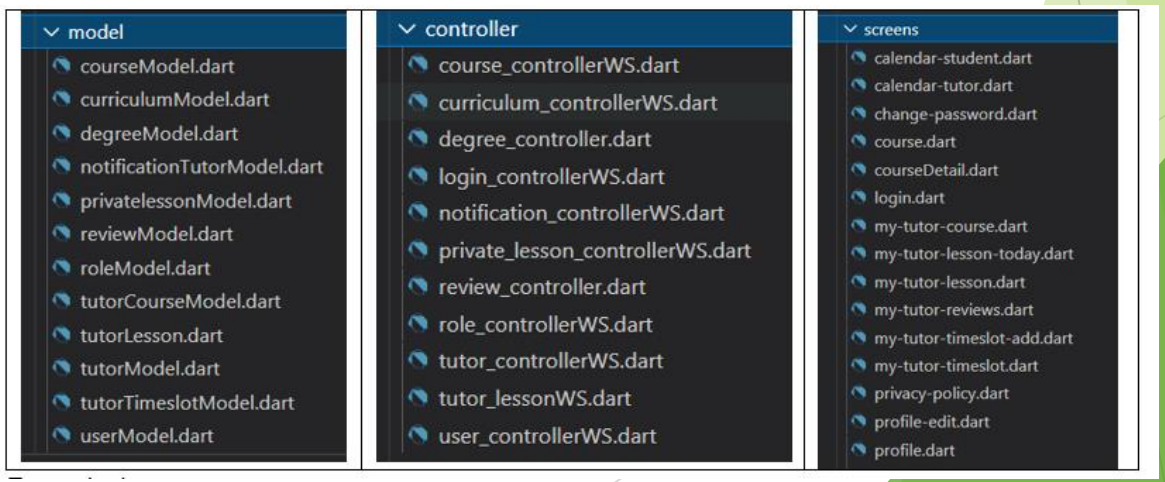
MySQL Database

Paradigma MVC (1)

```
CourseModel(
    this.course_id,
    this.course_name,
    this.course_cfu,
    this.enrollment_year,
    this.study_year,
    this.teaching_type,
    this.dac,
    this.department,
    this.curriculum,
    this.ssd,
    this.delivery_mode,
    this.language,
    this.didactic_period,
    this.component_type);
```

```
var course;
if (response.statusCode == 200) {
  // print(response.body);
  var courseJsonData = json.decode(response.body);
  course = CourseModel.fromJson(courseJsonData);
  // print(user);
}
return course;
```

- **Model**: classi che contengono i campi definiti nelle relative tabelle del DB: es. courseModel rispetta i campi definiti nella tabella course del DB;
- **Controller**:
  - si occupa di effettuare la **chiamata http**: chiama lo specifico WS per reperire i dati o effettuare l'operazione richiesta. Es. recuperare la lista dei corsi o effettuare l'iscrizione di nuovo utente;
  - mette i dati a disposizione della VIEW in modo che possa utilizzarli immediatamente: es. courseController mette a disposizione un oggetto di tipo CourseModel alla View CouseDetail in modo che possa rappresentare l'oggetto utilizzando una DataTable;
- **View**: si occupa di rappresentare i dati forniti dal controller (es. ListView o DataTable).



Paradigma MVC (2) - Esempio

https://www.e-tutoring-app.it/ws/course_list.php?course_id=3

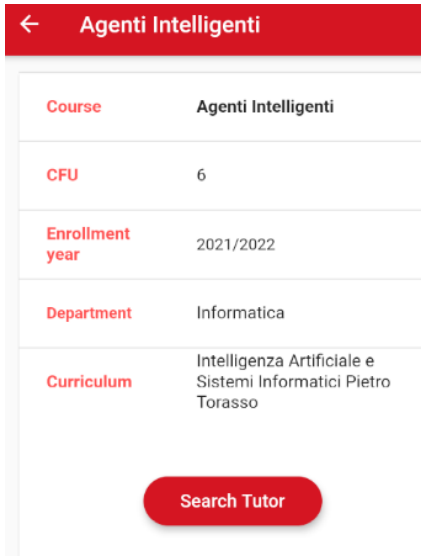
couse_detail.dart
StatefulWidget

- call controller method
- representing JSON objects in DataTable

course_controller.dart
getCourseDetailFromWS
REQUEST HTTP
WITH QUERY PARAMETER
(e. g. courseId)

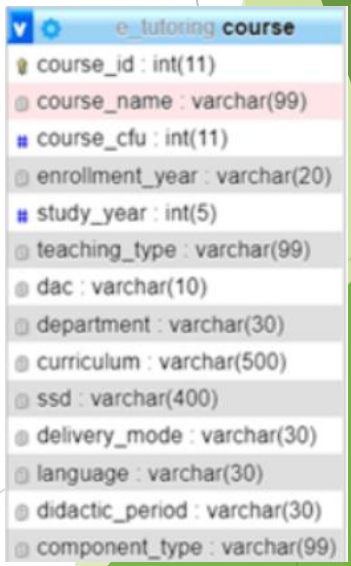
Use
courseModel.dart

Web Services
course_list.php
SELECT DISTINCT *
FROM course where
course_id = " .
$_GET['course_id']




```
{
  "course_id": "3",
  "course_name": "Agenti Intelligenti",
  "course_cfu": "6",
  "enrollment_year": "2021/2022",
  "study_year": "1",
  "teaching_type": "Caratterizzante",
  "dac": "MFN1348",
  "department": "Informatica",
  "curriculum": "Intelligenza Artificiale e Sistemi Informatici Pietro Torasso",
  "ssd": "INFORMATICA (INF/01)",
  "delivery_mode": "Convenzionale",
  "language": "Italiano",
  "didactic_period": "Secondo Semestre",
  "component_type": "Attività formativa monodisciplinare"
}
```


course
DB TABLE



CourseDetail VIEW - Esempio Rappresentazione dei dati nei Widget Utilizzo del FutureBuilder

courseDetail.dart

```
FutureBuilder<CourseModel>(
  future: getCourseDetailFromWS(http.Client(),this.courseData.course_id),
  builder: (BuildContext context, AsyncSnapshot<CourseModel> course) {
    List<Widget> children;
    if (course.hasData) {
      children = <Widget>[
        Container( color: Color.fromRGBO(205, 205, 205, 1),
            child: DataTable( dataRowHeight: 60,
            dataRowColor: MaterialStateColor.resolveWith((states) =>  Colors.white
              headingRowHeight: 0,
              columns: <DataColumn>[
                DataColumn( label: Text('', ),  ),
                DataColumn(label: Text( '', ), ),
              ], // <DataColumn>[]
              rows: <DataRow>[
                DataRow(
                  cells: <DataCell>[
                    DataCell(Text(
                      'Course',
```

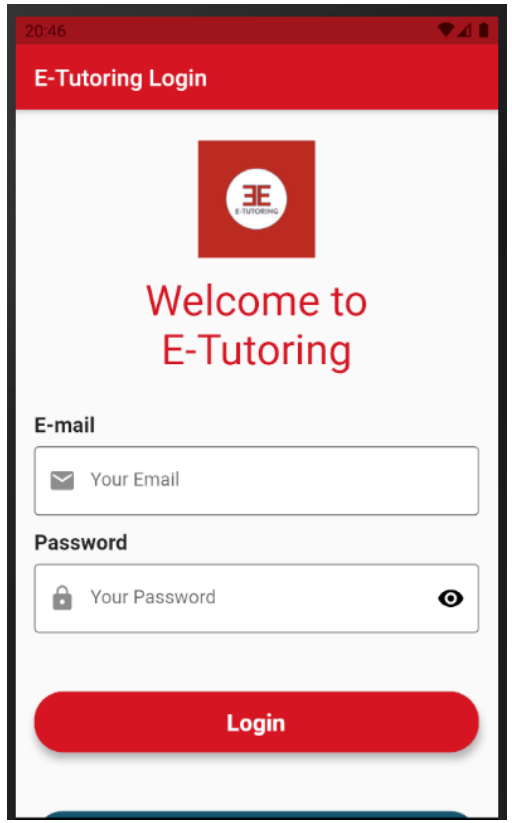
course_controller.dart

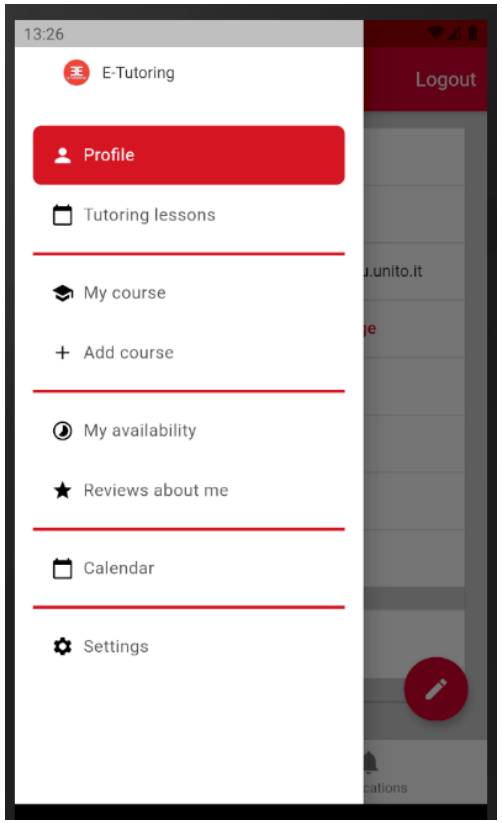
```
Future<CourseModel> getCourseDetailFromWS(
    http.Client client, String courseId) async {
  try {
    var queryParameters = {
      'course_id': courseId,
    };
    // print(queryParameters);
    var response = await client.get(
        Uri.https(
            authority, unencodedPath + "course_list.php", queryParameters),
        headers: <String, String>{'authorization': basicAuth});

    var course;
    if (response.statusCode == 200) {
      // print(response.body);
      var courseJsonData = json.decode(response.body);
      course = CourseModel.fromJson(courseJsonData);
      // print(user);
    }
    return course;
  } on Exception catch ($e) {
    print('error caught: ' + $e.toString());
    return null;
  }
}
```

Funzionalità dell’applicazione

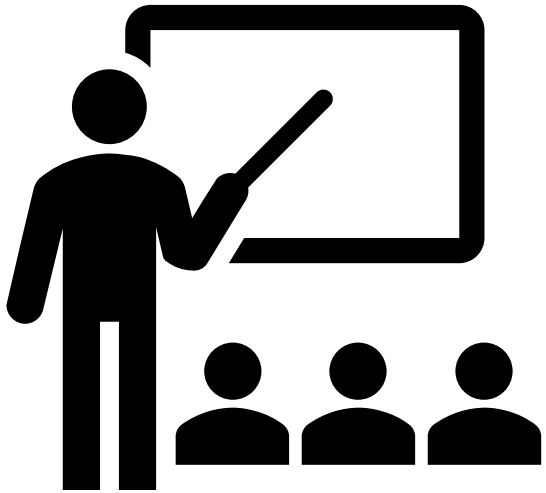
Funzionalità dell'Applicazione (generali)



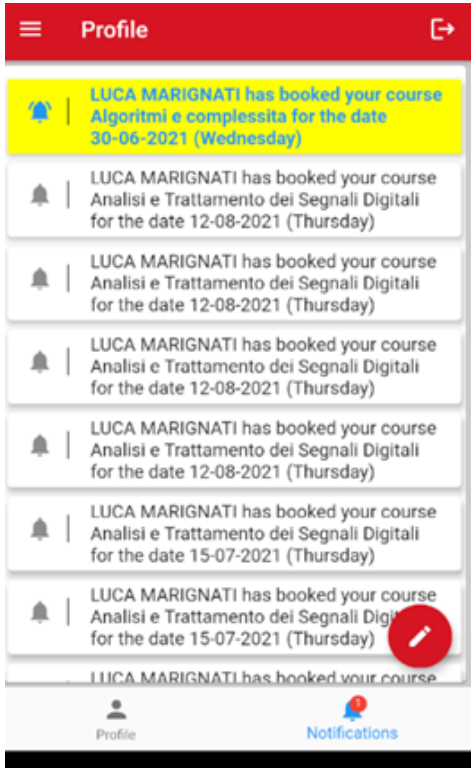


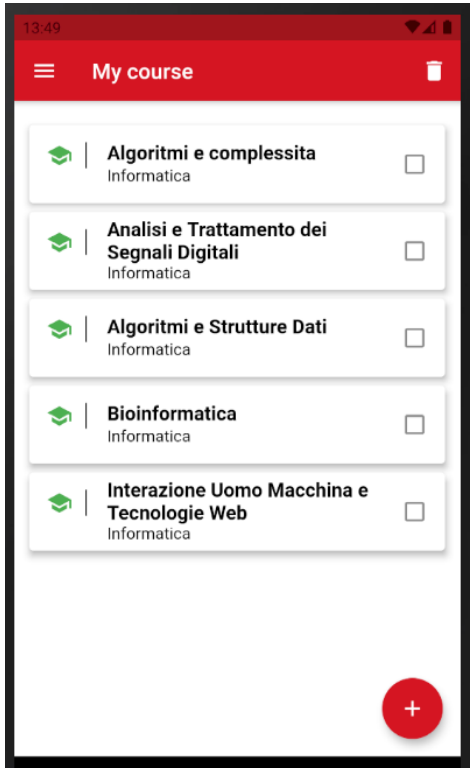
- Login
- Registrazione di un nuovo utente
- Privacy Policy
- Drawer menu
- Home (profilo e gestione notifiche)
- Edit profile
- Settings

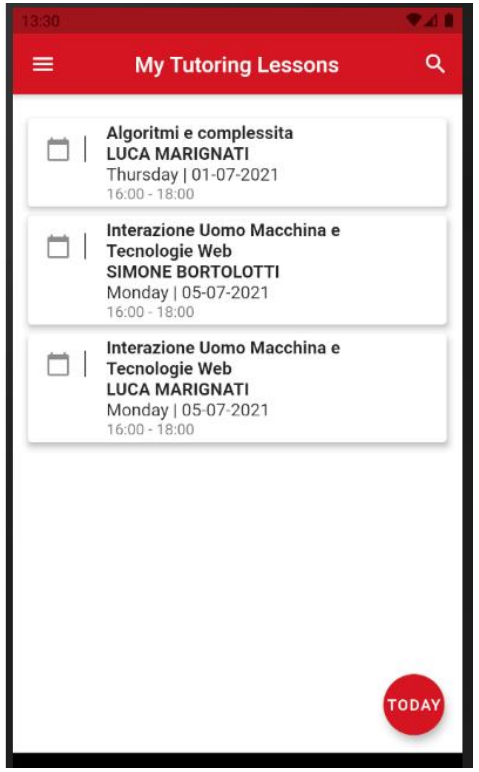
Funzionalità del Tutor (1)



- Visualizzazione delle lezioni private
- Calendario con visualizzazione delle lezioni Corsi (disponibili per tutoraggio)
- Possibilità di aggiungere un nuovo corso alla propria lista
- Disponibilità e possibilità di aggiungerne una nuova
- Visualizzazione delle recensioni ricevute

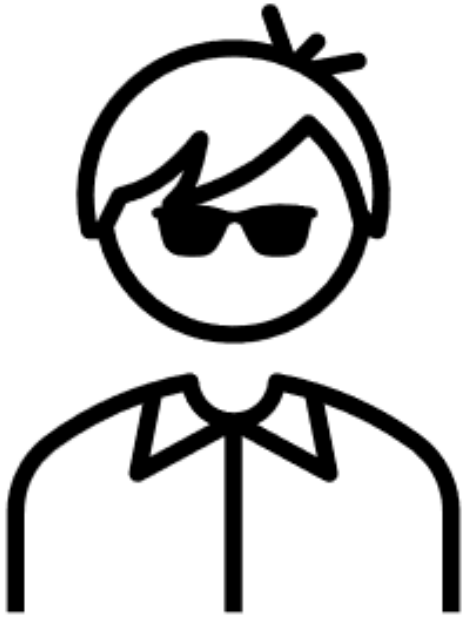




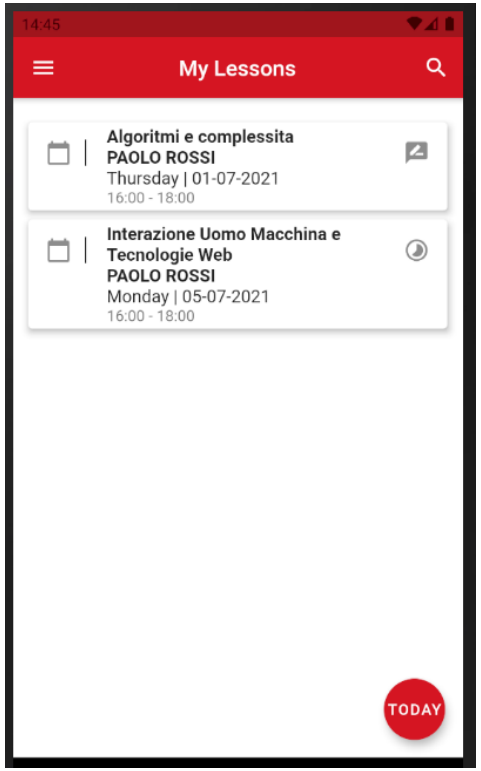


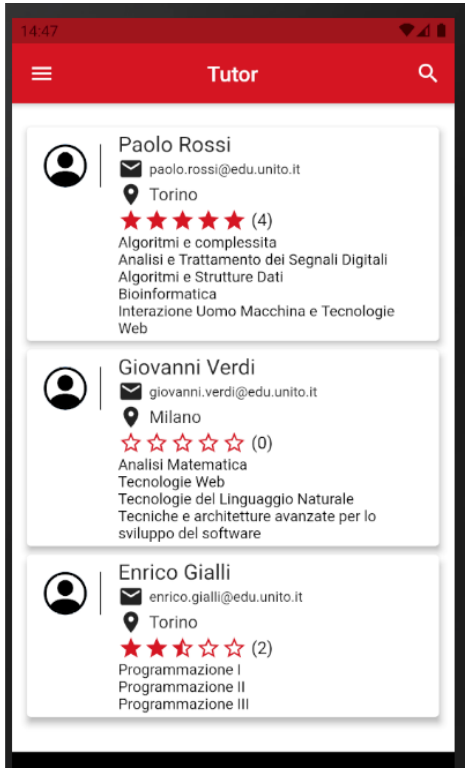
Funzionalità del Tutor (2) - Widget

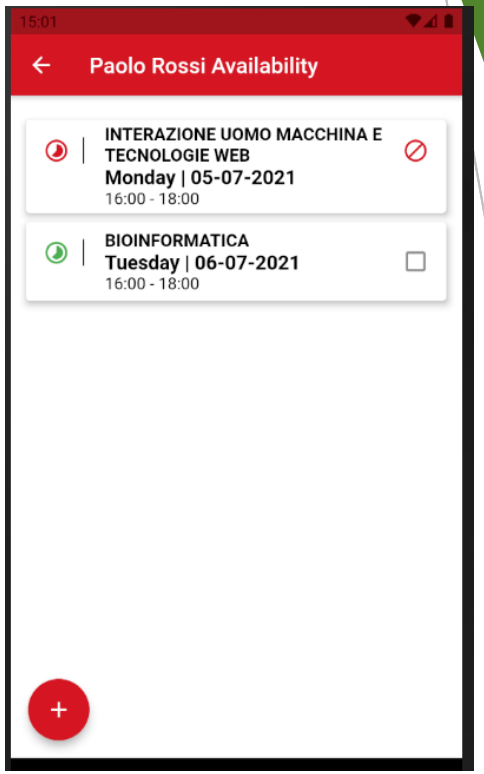
Funzionalità dello Studente (1)



- Visualizzazione di tutti i corsi a cui l'utente può prenotarsi
- Dettaglio del corso
- Visualizzazione delle lezioni prenotate
- Visualizzazione delle recensioni fatte
- Possibilità di aggiungere una recensione
- Possibilità di ricercare un tutor
- Visualizzazione del dettaglio del tutor
- Calendario







Funzionalità dello Studente (2) - Widget

Software

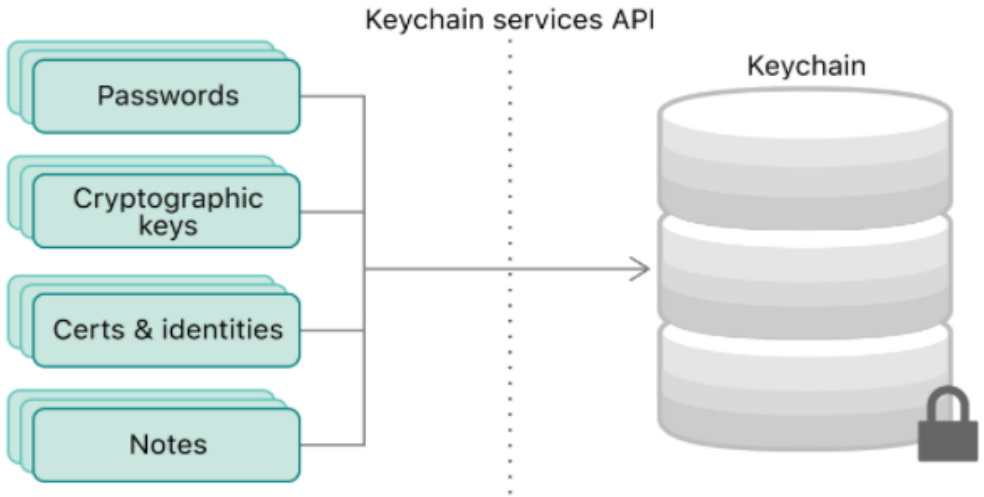
Software: caratteristiche principali

- **Flutter Secure Storage** per la memorizzazione delle credenziali dell'utente nel dispositivo;
- Sviluppo di widget e funzionalità differenti in base al **ruolo dell'utente** (Studente o Tutor);
- Supporto per le **lingue diverse**;
- Utilizzo di un Timer che esegue ogni 10 secondi una chiamata http per verificare la presenza di nuove prenotazioni da parte degli studenti (mostra una notifica al tutor);

Flutter Secure Storage

Flutter Secure Storage (1)

- È un plug-in Flutter per archiviare i dati in maniera sicura;
- permette di **astrarre dalle primitive native** dei vari dispostivi fornendo dei metodi che permettono di scrivere (WRITE) e leggere (READ) chiavi:
  - write(key, value); read(key); delete(key).
- abbiamo utilizzato questa libreria per l'archiviazione sicura delle informazioni dell'utente (username, password e ruolo) nel dispositivo;
- implementazione: è la stata implementata la classe **UserSecureStorage** che si occupa di comunicare con il KeyStore (per Android) o il Keychain (per iOS).



Flutter Secure Storage (2)

```dart
import 'package:flutter_secure_storage/flutter_secure_storage.dart';

class UserSecureStorage {
  static final _storage = FlutterSecureStorage();

  static const _keyEmail = 'email';
  static const _keyPassword = 'password';
  static const _keyRole = 'role';

  static Future setEmail(String email) async =>
      await _storage.write(key: _keyEmail, value: email);

  static Future<String> getEmail() async => await _storage.read(key: _keyEmail);

  static Future setPassword(String password) async =>
      await _storage.write(key: _keyPassword, value: password);

  static Future<String> getPassword() async =>
      await _storage.read(key: _keyPassword);

  static Future setRole(String role) async =>
      await _storage.write(key: _keyRole, value: role);

  static Future<String> getRole() async => await _storage.read(key: _keyRole);

  static void delete(String key) async => await _storage.delete(key: key);
}
```

Ruolo dell'utente (Studente/Tutor)

Ruolo dell'utente (1)

- In fase di login, oltre a e-mail e password, viene salvato nel Secure Storage (metodo **UserSecureStorage.setRole(role)** anche in ruolo in modo che possa essere disponibile in tutti i Widget;
- https://www.e-tutoring-app.it/ws/get_user_role.php?email=davide.decenzo@edu.unito.it

```
// get user role
dynamic role = await getRoleFromWS(http.Client(), emailController.text);
await UserSecureStorage.setRole(role.role_name);
```

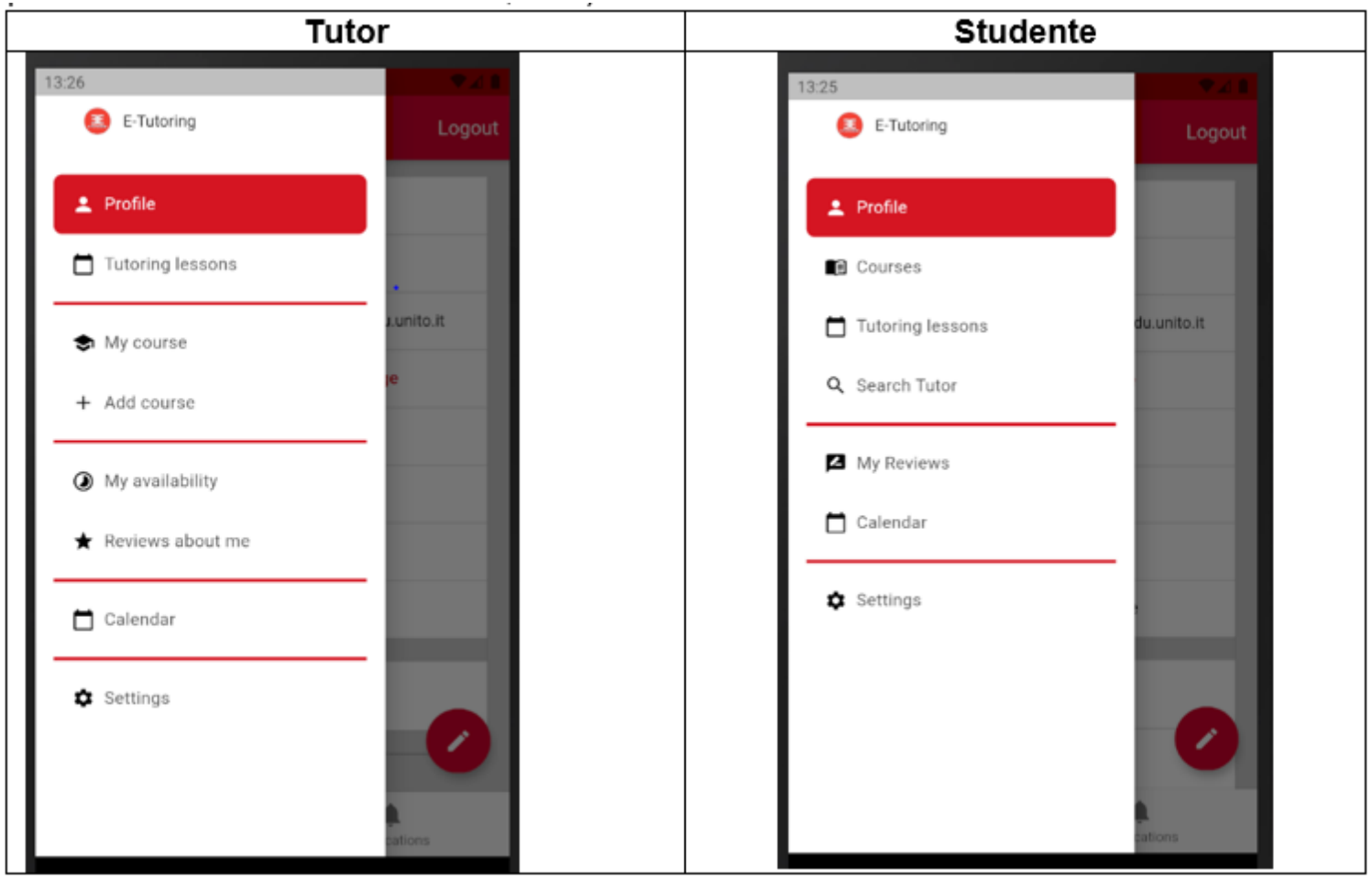
- in fase di visualizzazione del Widget viene recuperato il ruolo dal Secure Storage (metodo **UserSecureStorage.getRole()**);

```
children = <Widget>[
  new FutureBuilder<String>(future: UserSecureStorage.getRole(),
      builder: (BuildContext context, AsyncSnapshot<String> role) {
        switch (role.connectionState) {
          case ConnectionState.none:
```

- Utilizzo: per gli utenti con ruolo «Studente», per esempio, viene visualizzata la matricola

```
if (role.data == "Student")
  DataRow(cells: <DataCell>[
      DataCell(Text(AppLocalizations.of(context).number, st
      DataCell(Text("${user.data.badge_number}", style: Tex
    ], // <DataCell>[]
  ), // DataRow
```

Ruolo dell’utente (2)
DIFFERENZIAZIONE DEL DRAWER MENU



Supporto per lingue diverse

Supporto per lingue diverse (1)

- Lingue supportate dell'applicazione: inglese e italiano;
- l'applicazione adatta i propri contenuti in base alla lingua scelta sfruttando le librerie **flutter_localizations** e **intl** che forniscono le funzionalità di internazionalizzazione, localizzazione e la traduzione dei messaggi;
- la **cartella l10n** contiene i file di traduzioni in formato ARB in cui le risorse sono codificate come oggetti JSON (chiave-valore);

app_en.arb

```
"welcome": "Welcome to\nE-Tutoring",
"@welcome": {
  "description": "Welcome to E-Tutoring"
},
"login": "Login",
"@login": {
  "description": "Login"
},
"signup": "Sign up",
"@signup": {
  "description": "Sign up"
},
"yourpassword": "Your Password",
"@yourpassword": {
  "description": "Your Password"
},
"youremail": "Your Email"
```

app_it.arb

```
"welcome": "Benvenuto in\nE-Tutoring",
"login": "Accedi",
"signup" : "Registrati",
"yourpassword": "Inserisci la password",
"youremail": "Inserisci l'email",
"error_email_empty": "Inserisci la tua email",
"error_email_not_valid": "Inserisci un email valida",
"error_password_empty": "Inserisci la tua passowrd",
"error_password_not_valid" : "Inserisci una password
"confirmPassword" : "Conferma Password",
"passwords_not_match": "Le password inserite non comb
"select_degree_course" : "Scegli il corso di laurea",
```

Supporto per lingue diverse (2) – utilizzo

- Si utilizza la libreria AppLocalizations definendo la chiave della traduzione (es. login):

```
appBar: AppBar(
  title: Text("E-Tutoring " + AppLocalizations.of(context).login),
  backgroundColor: Color.fromRGBO(213, 21, 36, 1),
  actions: [LanguagePickerWidget()],
), // AppBar
backgroundColor: Colors.white,
body: Scaffold(
```

Timer per la gestione delle notifiche

Timer per la gestione delle notifiche (1)

Il timer esegue ogni 10 secondi la chiamata http che restituisce la lista delle notifiche relative al tutor (tutor: es. paolo.rossi@edu.unito.it): vengono mostrate le prenotazioni degli studenti ai corsi sostenuti dal tutor.
https://www.e-tutoring-app.it/ws/notifications_tutor.php?email=paolo.rossi@edu.unito.it

Es.

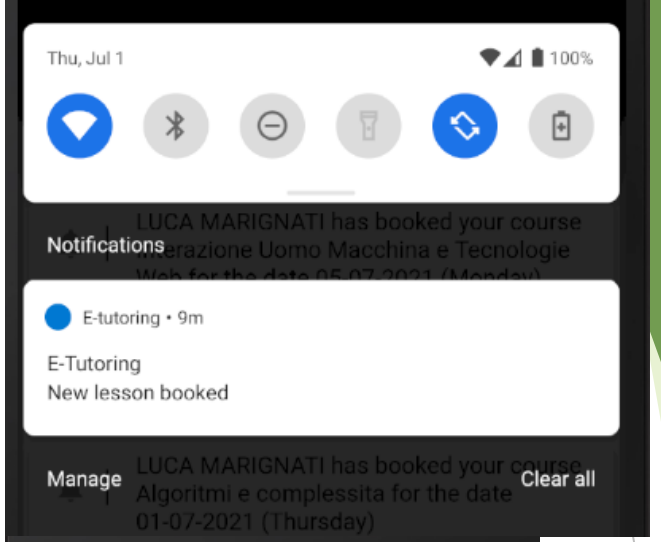
- tempo t, notificationList è un array di lunghezza 3
- tempo t + 10 secondi: newNotificationsList è una array di lunghezza 4
- 3 < 4 → vuol dire che c'è stata una nuova prenotazione → notifica
- showNotification (FIREBASE) → **notifica a livello globale** SO;

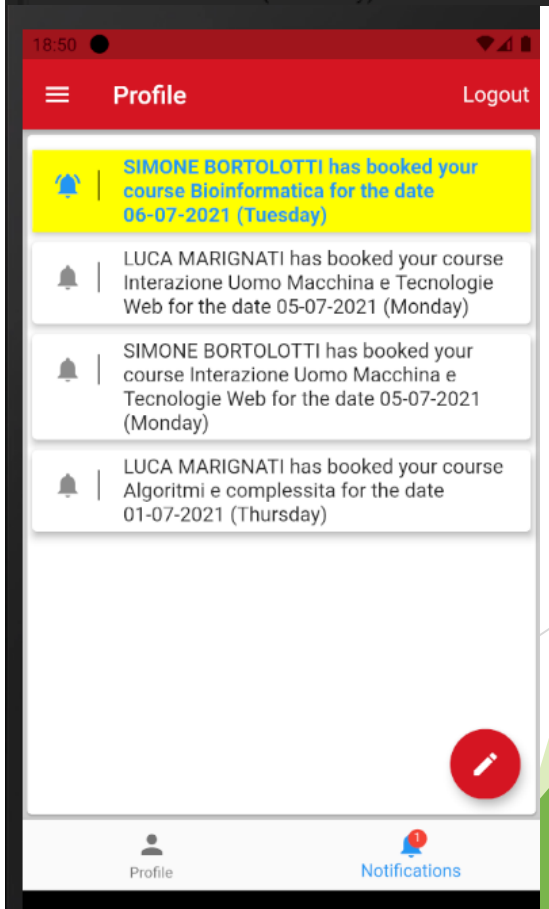
FIREBASE: https://console.firebase.google.com/u/0/project/flutterpushnotification-1e340/notification

- **notifica a livello locale** dell'applicazione (setState): aggiornamento lista e badge;

Timer (2)
codice + notifiche globali e locali

```
timer = Timer.periodic(Duration(seconds: 10), (Timer t) {
  getNotificationsTutorFromWS(http.Client())
      .then((newNotificationsList) => {
            // print(newNotification),
            print(newNotificationsList.length),
            print(notificationsList.length),
            if (notificationsList.length >= 0 &&
                notificationsList.length < newNotificationsList.length)
              {
                showNotification(),
                setState(() {
                  notificationsList = newNotificationsList;
                  for (var notification in notificationsList) {
                    if (notification.check == "0") {
                      badgeNotificationNumber++;
                    }
                  }
                })
              }
          });
}); // Timer.periodic
```





Timer (3)
Web Services di gestione delle notifiche check field

Di particolare importanza è il campo «check» restituito dal WS in quanto specifica se il tutor ha visionato o meno la notifica:

- check = 0: il tutor non ha visto la notifica (nella view/widget questo è evidenziato dal colore giallo della nuova notifiche – vedi slide precedente);
- check = 1: il tutor ha visto la notifica.

```
selected: notification.check == "0" ? true : false,
selectedTileColor: notification.check == "0" ? Colors.yellowAccent : Colors.white,
```

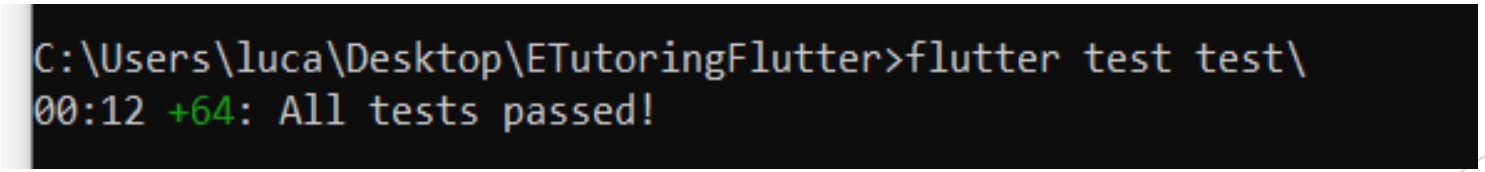

```
{
  "notifications_tutor_id": "47",
  "private_lesson_id": "69",
  "date": "2021-07-04 16:36:18",
  "check": "0",
  "user_id": "11",
  "tutor_course_id": "77",
```

```
{
  "notifications_tutor_id": "46",
  "private_lesson_id": "68",
  "date": "2021-07-01 15:08:07",
  "check": "1",
  "user_id": "11",
  "tutor_course_id": "78",
```

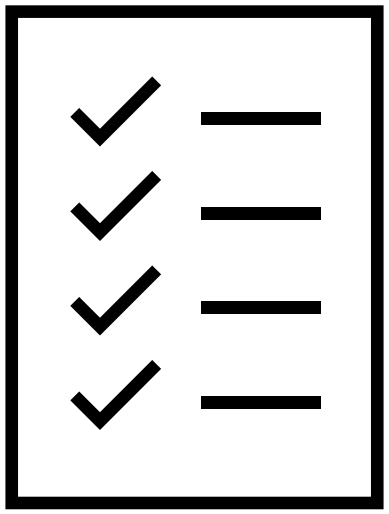
Testing

Testing (1)

- Abbiamo sviluppato i test suddividendo in cartelle rispettando la struttura del codice in modo da avere un'esatta corrispondenza tra implementazione e test:
  - nella cartella di "*test/controller*" abbiamo testato le funzioni relative a "*lib/controller*" (in particolare le chiamate http);
  - nella cartella "*test/screens*" abbiamo testato le funzioni relative a "*lib/screen*";
  - nella cartella "test/utils" abbiamo testato le funzioni relative a "*lib/utiils*";
  - nella cartella "*test/widgets*" abbiamo testato le funzioni relative a "*lib/widgets*".



Testing (2) – test/controller - Mockito



- Utilizzando la libreria **Mockito** abbiamo testato i metodi relativi alle chiamate http.
- Per ogni metodo del controller abbiamo implementato un test contenente 2 casi:
  - risposta HTTP con status 200: operazione va a buon fine;
  - risposta HTTP con status 404 (NOT FOUND): operazione fallita;

test/controller (3) – status 200 esempio: getCurriculumListFromWS

```dart

getCurriculumListFromWSTest() {
  Run | Debug
  group('getCurriculumListFromWS', () {
    Run | Debug
    test(
        'return a List of CurriculumModel if the http call completes successfully',
        () async {
      final client = MockClient();
      var queryParameters = {
        'degree_name': 'informatica',
        'degree_type_note': 'Laurea Magistrale'
      };
      when(client.get(
              Uri.https(
                  authority,
                  unencodedPath + "curriculum_path_by_degree.php",
                  queryParameters),
              headers: <String, String>{'authorization': basicAuth}))
          .thenAnswer((_) async => http.Response(
              '[{"degree_path_name": "Immagini, Visione e Realtà Virtuale"}]',
              200));

      List<CurriculumModel> curriculumList = await getCurriculumListFromWS(
          client, 'informatica', 'Laurea Magistrale');
      expect(curriculumList, isA<List<CurriculumModel>>());
    });
  });

```

test/controller (4) – status 404 Not Found esempio: getCurriculumListFromWS

```
Run | Debug
test(
    'return an empty List of CurriculumModel if the http call completes with fails: erro
    () async {
  final client = MockClient();
  var queryParameters = {
    'degree_name': 'informatica',
    'degree_type_note': 'Laurea Magistrale'
  };
  when(client.get(
          Uri.https(
              authority,
              unencodedPath + "curriculum_path_by_degree.php",
              queryParameters),
          headers: <String, String>{'authorization': basicAuth}))
      .thenAnswer((_) async => http.Response(
          '''[{"degree_path_name": "Immagini, Visione e Realtà Virtuale"},
          {"degree_path_name": "Reti e Sistemi informatici"}]''', 404));

  List<CurriculumModel> curriculumList = await getCurriculumListFromWS(
      client, 'informatica', 'Laurea Magistrale');
  expect(curriculumList, []);
});
}
```

Conclusioni

Conclusioni (1) – Native Application

Vantaggi	Svantaggi
Applicazioni ottimizzate per uno specifico sistema e garanzia di **prestazioni ottimali**. Rendering grafico accelerato. Le interfacce utente sono su misura per la piattaforma di destinazione (Android o iOS) e quindi sono scorrevoli e piacevoli da usare;	**Portabilità**: nessuna funzionalità multi-piattaforma.
Nessun limite nella realizzazione: pieno controllo sul dispositivo e i suoi sensori. Vi è la possibilità completa di interfacciamento con le API ((hardware e software) della piattaforma, es. accesso a CAMERA e GPS.	**Conoscenza specifica del linguaggio nativo** (es. Swift, Java o Kotlin): sono richiesta maggiori competenze per lo sviluppo e una **curva di apprendimento più ripida e difficoltosa.**
Elevata visibilità nel relativo App Store: il Play Store di Android tiene maggiormente in considerazione le app sviluppate per lo specifico sistema operaitivo.	**Cicli di sviluppo più lunghi**.

Conclusioni (2) – Cross-platform

Vantaggi	Svantaggi
Portabilità: unico codice per tutte le piattaforma (iOS e Android).	Alcune funzionalità sono difficilmente accessibili senza usare codice nativo: **potrebbe essere necessaria un'integrazione nativa** (es. cloud messagging per iOS e Android). E necessaria una conoscenza media delle piattaforme di sviluppo (le parti non condivise sono scritte in codice nativo).
Manutenibilità: realizzando un unico codice è più veloce mantenerlo (es. aggiornamento librerie), correggerlo e identificare malfunzionamenti.	**Utilizzo di Plugins e dipendenza da terze parti**: le soluzioni multipiattaforma moderne sono in grado di accedere a quasi tutte le funzionalità native di un dispositivo tramite l'utilizzo di Plugins. **L'uso di questi plugin aggiunge complessità e dipendenze allo sviluppo. Ricorrere a codici diversi e di terze parti comporta un minor controllo sulla qualità e sulla flessibilità.**
Facilità e velocità di sviluppo (tempi di rilascio brevi). Abbassamento dei tempi di realizzazione e garanzia di **buone prestazioni** (risparmio nello sviluppo).	Dipendenza da un framework e possibilità instabilità dello stesso Framework (es. presenza di vulnerabilità relative alla sicurezza).
Curva di apprendimento meno ripida. Il linguaggio risulta più facile da capire.	Prestazioni e user-experience inferiori rispetto ad un'applicazione nativa.
L'utente medio non nota la differenza tra un app nativa e un app cross-platform.	

Conclusioni (3) – Dart

Linguaggio object-oriented + programmazione funzionale	Garbage Collector (come in Java)	Type safe	Doppio controllo dei tipi: AOT (Ahead Of Time) + JIT (Just in Time)
Modello a Single Thread & Event Loop ma con possibilità di scrivere codice concorrente	Tipo DYNAMIC per rappresentare un tipo di dato dinamico a runtime	Programmazione asincrona e Future	ASYNC E AWAIT

Conclusioni (4) - Flutter

Cross-platform e indipendenza dal sistema operativo

Linguaggio di programmazione con una curva di apprendimento veloce, in quanto orientato agli oggetti e con molte similitudini a Java

Sviluppo rapido, utilizzando la stessa codebase per più sistemi operativi, è possibile riutilizzare diverse parti di codice.

Performance simili a quelle native

Hot reloading – impatto positivo sulle tempistiche di sviluppo

Riduzione dei costi di sviluppo

Linguaggio Dart, facile da usare, veloce da sviluppare e flessibile

Widget personalizzabili

Framework open source

Grande community sviluppatori legata a Google

Conclusioni (5) – Differenze e analogie con Android

	Android	Flutter
UI ASINCRONA	Quando si vuole eseguire codice molto pesante o che richiede tempo (es. chiamata di rete) si sposta il lavoro su un thread in background in modo da non bloccare il thread principale (AsyncTask e Service).	L'event loop di Flutter è equivalente a quello di Android (thread principale).
VIEW	Tutto ciò che compare a schermo è una View. Per definire il layout a una View è necessario scrivere un file .xml che contiene tutte le caratteristiche del layout stesso	La corrispettiva astrazione delle View in Flutter sono le Widget.
INTENTI	In Android vengono dichiarati nel file AndroidManifest.xml e vengono utilizzati per navigare tra le Activities e comunicare con i componenti.	In Flutter si utilizzano i Navigator e le Route per navigare tra le schermate. Le informazioni di layout vengono specificate dal widget singolarmente man mano che vengono modellate per essere più performante ed intuitivo.
ATTIVITÀ E FRAMMENTI	Le Activities rappresentano una singola azione che l'utente può compiere. Mentre un Fragment rappresenta un comportamento o una parte dell'interfaccia utente;	in Flutter, entrambi questi concetti ricadono nei Widget.
GRADLE	In Android, è possibile aggiungere dipendenza all'applicazione inserendo lo script necessario all'interno del file Gradle;	in Flutter, si utilizza il gestore di pacchetti Pub e le dipendenze vengono inserite nel file pubspec.yaml.

Grazie per l’attenzione